Towards a Sustainable IoT with Last-Mile Software Deployment

Rustem Dautov
SINTEF Digital
Forskningsveien 1, 0373 Oslo, Norway
Email: rustem.dautov@sintef.no

Hui Song
SINTEF Digital
Forskningsveien 1, 0373 Oslo, Norway
Email: hui.song@sintef.no

Nicolas Ferry
Université Côte d'Azur
I3S/INRIA Kairos
Sophia Antipolis, France
nicolas.ferry@univ-cotedazur.fr

***Abstract*—The ubiquitous presence of billions of sensor-enabled IoT devices actively contributes to the generation of extreme amounts of electronic waste. To a great extent, this is caused by the low cost and short lifespan of electronic components, as a result of which it is often more convenient for consumers to buy a new device instead of re-using or recycling the old one. With the increased computing and connectivity capabilities, however, individual IoT devices are already capable to communicate with a local network gateway to receive code updates and thus extend their lifespan. This paper describes how these capabilities can be exploited to enable last-mile software deployment at scale. We propose a hierarchical architecture, in which software updates from the cloud are provisioned to terminal devices via edge gateways in a scalable and targeted manner. By enabling such an end-to-end software deployment architecture, the approach promotes hardware re-use and re-purposing and thus contributes to the creation of a more sustainable IoT.**

***Index Terms*—Internet of Things, Edge Computing, Sustainable IoT, Last-Mile Software Deployment, Firmware Updates**

I. Introduction

Billions of sensors are already embedded in a vast array of networked physical objects, and soon it will be difficult to buy a new product without some kind of connected smarts. Controlling an appliance from a mobile phone is already a standard feature and ubiquitous sensors and actuators will transform industrial automation, buildings, and our interaction with the surrounding world. This global digital transformation is driven by a new class of microcontrollers (MCUs), i.e. stand-alone miniature computing chips equipped with flash memory for code storage, and a small amount of RAM in which to execute. They are also able to communicate with hardware peripherals via General Purpose Input/Output (GPIO). This includes both digital I/O and associated protocols such as I2C, Serial, SPI, and CAN, as well as analog I/O for reading data from environmental sensors and other analog sources. MCUs are not designed to run heavy OSs such as Linux or Windows that are based on a multi-user, multi-application, hardware-abstracted paradigm, in which the OS is often the heaviest consumer of resources. Instead, MCUs are typically either OS-less, or running Micro Real-Time Operating System (µRTOS) which is just enough to run the application, providing it full, direct and low-latency access to the chip hardware. This power efficiency means that MCUs, even with sensors, can run for years on a small battery, or indefinitely by adding energy harvesting units such as a small solar cell. By using a fraction of the power that single-board computers do, their total cost of ownership is significantly lower. The small power draw also means that they can be powered by alternative sources than a wall plug, which means they can be placed virtually anywhere.

With the rapid development of the Internet of Things (IoT), MCU-enabled tiny sensor devices have penetrated almost every aspect of people's everyday life. This rising *sensor-based economy*, however, is actively contributing to the e-waste problem, as it also produces myriads of obsolete devices with much shorter lifespans and offers no way of properly disposing them [1]. The United Nations estimated 74.7 million tonnes of e-waste to be generated annually by 2030, thus almost doubling in only 16 years, with less than a third of it being recycled [2].

While MCUs have been embedded in everything from toys to cars for several decades now, it is only been in the last few years that they have become truly useful for the IoT. They are not only sufficiently powerful to run complex applications, but many have also received built-in support for gateway connectivity such as Ethernet, WiFi, Bluetooth, and others. Many of them have also opened up their memory buses to be extended with external flash and RAM. As we further argue in this paper, these increased connectivity capabilities can and should be exploited in order to extend the lifespan of IoT devices and reduce the e-waste generation rates. More specifically, the main contribution of this paper is the scalable hierarchical agent-based approach to uploading code to terminal IoT devices via edge gateways. The proposed architecture enables re-programming and re-purposing of devices, not immediately connected to the Internet, thus contributing to the creation of a more sustainable IoT.

The rest of the paper is organised as follows. Section II discusses the motivation behind the presented work by introducing the research context, outlining the limitations of the existing research efforts, and formulating the problem. Section III presents the proposed approach by describing the conceptual architecture of the last-mile deployment agent and the proof of concept implementation. Section IV discusses the results and closes the paper with some concluding remarks.

This is the author accepted version of an article published in
2021 IEEE Symposium on Computers and Communications (ISCC)
https://doi.org/10.1109/ISCC53001.2021.9631250

II. Research Context, Related Works, and Problem Statement

The increasing number of embedded electronic equipment led to the increasing amount of electronic waste (or simply *e-waste*), i.e. any electrical or electronic equipment that has been discarded at any step of its lifecycle. E-waste is particularly dangerous due to toxic chemicals that naturally leach from the metals inside when buried. The e-waste problem is recognised at the highest governmental levels leading to the initiatives such as EU's Waste from Electrical and Electronic Equipment (WEEE) Directive[1] and US's International E-Waste Management Network (IEMN).[2] Both initiatives bring together various stakeholders and policy makers to join efforts on e-waste reduction aiming to achieve sustainable circular economy.

A. E-Waste and Green IoT

The most hazardous part of WEEE are printed circuit boards (PCB) [3]. They contain flame retardants, precious metals and multiple electronic components of different shape and size soldered to the board (e.g. resistors, inductors, capacitors, integrated circuits). The presence of multiple elements of different nature requires differentiated treatment, which makes recycling WEEE especially challenging and costly. Briefly, WEEE recycling can be divided into two main categories – namely, *physical* and *chemical*. The former involves physical dismantling of electronic components (and in rear occasions – replacing faulty elements for repairing and further reuse), while the latter handles the actual recycling of disassembled PCB elements and polymers using chemical processes.

Another negative aspect of the ubiquitous IoT is the increased energy consumption, which has been the main research focus for the *Green IoT* community who aim at reducing energy consumption of sensor devices to achieve sustainable operation of large-scale IoT systems [4]. Even though modern devices consume less and less power individually, their combined and continuously growing number still outweighs the benefits.

In recent years, the Green IoT concept has expanded beyond energy efficiency into a wide range of related solutions contributing to sustainable and environmentally-friendly IoT systems in general. More specifically, apart from the physical and chemical recycling, the research community has been looking into possible alternative approaches to e-waste reduction by creating dissolvable electronics using safe and environmentally friendly materials [5], [6], thus aiming at the very core problem of e-waste, i.e. toxic chemicals leaching into the environment. Although still too expensive to produce, dissolvable electronics may prove to be useful in the near future with the current pace of IoT development.

A potential alternative to the conventional recycling is *hardware reuse and re-purposing*, which regrettably has not gained much research attention yet. The main idea behind this approach is to opt for updating IoT software on idle and dormant devices, whenever possible, to be used in a new application scenario or another IoT system. This would prevent wasted electronics from ending up in the environment and also reduce resource-intensive and expensive physical/chemical recycling operations. Moreover, increased energy consumption by IoT devices can often be caused by outdated or misconfigured firmware, which has not been properly updated in time. For example, some weather sensors might have been hard-coded at the production phase to probe the environment every second, while in practice a one-minute interval would be sufficient for most application scenarios.

B. Related Works

The start-of-the-art Infrastructure as Code (IaC) tools automate the deployment of one application on one device, or a predefined set of devices, but lack the support for scalable distribution of multiple variants across a large fleet. They also do not provide sufficient automated support for updating devices with constrained resources and limited (or none) Internet connectivity [7]. This latter aspect has become particularly important in the recent years with the emerging need to update firmware (low-level executable code deployed on MCU-enabled devices) on terminal IoT devices. Such embedded and MCU-enabled devices traditionally have been flashed with 'one-off' firmware not to be updated in the future, This was due to hardware and connectivity constraints, as well as security considerations, which limited consequent manipulation of embedded code once a device leaves the production line, is shipped to a customer and is finally put into operation. The situation has only started to change recently. Increasingly capable IoT devices are now seen as active contributors to the common pool of shared computing resources, which can be iteratively assigned and deployed with updated firmware. This has also led to the so-called concept of IoT-Edge-Cloud computing continuum, where computing and storage tasks are distributed across all three levels.

Provisioning new firmware to terminal IoT devices via an Internet-connected gateway or a smartphone was a natural fit. This way, IoT devices connected to a gateway via a non-TCP/IP network interface which supports a firmware flashing protocol (e.g. Bluetooth or a serial port) can receive code updates. Examples of such approaches can be found in [8], [9], [10]. In parallel to this, some recent research works also investigated how narrow-bandwidth networks can be used to enable over-the-air firmware updates for battery-powered devices [11], [12], [13].

The run-time software management in the IoT is closely related to the concept of software-defined architectures. The term *software-defined IoT* has been coined, albeit there is no common agreement on the actual concept yet. Most research work focus on the software-defined networks in the IoT systems [14], [15], [16], [17], while some other focus on elastic provisioning of cloud resources for IoT systems [18]. As far as the actual IoT hardware re-purposing and re-programmability are concerned, there appears to be very few approaches. A

[1] https://ec.europa.eu/environment/topics/waste-and-recycling/waste-electrical-and-electronic-equipment-weee_en

[2] https://www.epa.gov/international-cooperation/international-e-waste-management-network-iemn

representative study is reported in [19], where authors look at the challenge of re-using existing IoT infrastructure in a university campus from several perspectives, i.e. technical, economical, environmental, legal, etc. Albeit very relevant, the discussion remains at a high level and lacks technical details of the implemented solution.

C. Problem Statement

While none of the cited research works specifically targets the sustainability problem in the IoT, they demonstrate how firmware updates on IoT devices can be managed within some specific ad-hoc scenarios. Individually they can be seen as building blocks for creating an end-to-end pipeline for software management across the whole IoT-Edge-Cloud continuum. Indeed, what was hardly possible just a decade ago, has become technically feasible today with the advances in electronics design and manufacturing. Albeit still relatively resource-constrained compared to conventional single-board computers, MCU-based IoT devices are becoming more and more capable in terms of networking and computing resources, enabling remote access and re-programmability. What is still missing, however, is a scalable orchestration layer between the edge and the IoT layers that would be able to communicate with the centralised cloud via the Internet – on the one hand, and with downstream IoT devices via a local connection – on the other. Such a bridge would enable global access to locally connected IoT devices via nearest edge gateways, and thus facilitate automated and scalable firmware deployment.

We refer to this problem as the *last-mile software deployment*, i.e. deployment of code to terminal IoT devices, which do not have immediate connection to the Internet, yet are able to communicate with a local edge gateway. The presented approach is part of a wider research effort, which applies model-driven engineering techniques to automate software management activities across the IoT-Edge-Cloud computing continuum [20], [21].

III. Proposed Approach

In a simplified IoT hierarchy, we distinguish between three main elements, i.e. a centralised cloud platform, Internet-connected edge gateways, and downstream IoT devices not connected to the Internet, but only to a local edge gateway. Accordingly, data communication takes place via two links between *i)* the cloud platform and edge gateways and *ii)* edge gateways and IoT devices. While both types of network interaction are a common state of practice, it is still a challenge to propagate data (e.g. firmware updates) from the cloud down to terminal IoT devices via edge gateways in a generic, scalable, and automated manner.

A. Last-Mile Software Deployment: Conceptual Architecture

The cloud counterpart is only able to communicate with edge gateways, and is usually not aware of the billions of terminal devices installed in the field. In these circumstances, it is typically not feasible to target a firmware update to a specific single IoT device (using, for example, its ID). Instead, it is required to annotate firmware updates with *target conditions* and push them down to the edge layer, which, in its turn, will evaluate the conditions and route the updates to matching IoT devices in its subnet. Thus, edge gateways become the key element in this software management hierarchy [22]. Accordingly, we tackle the challenge of last-mile software deployment by introducing a *context-aware deployment agent* to be installed on edge gateways. The main role of this agent is to bridge the communication gap between the cloud and IoT devices by 'routing' software updates to target IoT devices taking into account the current cyber-physical context. The deployment agent is expected to be operating in a daemon-like manner, implementing the following four-fold functionality, as depicted in Fig. 1:

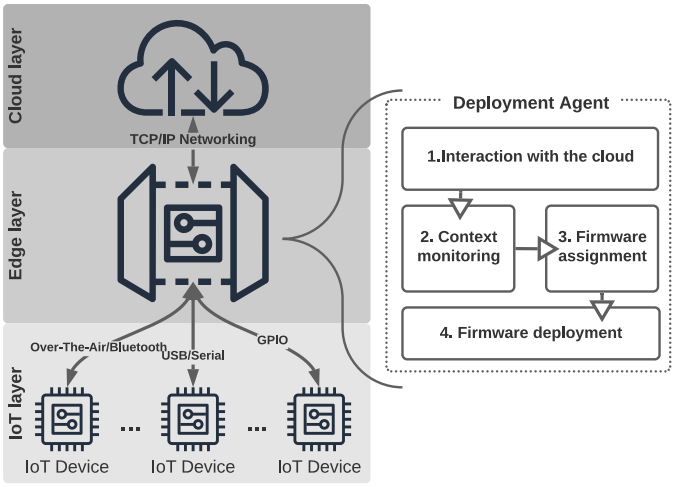

Fig. 1. Last-mile software deployment agent.

1) **Interaction with the cloud**: edge gateways sit in the middle of the IoT hierarchy and are able to receive updates from the cloud. It is expected that the cloud platform maintains a list of registered edge gateways and communicates with them asynchronously using standard pub-sub mechanisms or – less usual – synchronously using direct method invocations. In both cases, edge gateways should be able to receive update commands (annotated with target conditions) along with the actual code to be deployed on IoT devices.

2) **Context monitoring**: edge gateways are expected to perform continuous context monitoring by keeping track of associated downstream devices. These are closely located IoT devices using the gateway for pushing collected sensor data via one of the available communication interfaces, such as Bluetooth, ZigBee, serial USB connection, or even direct physical wiring via GPIO pins. The collected context information about downstream IoT devices, among other things, may include the current firmware version, the manufacturer brand and model, MCU architecture, available physical interfaces, installed hardware sensors/actuators, etc. It is important to collect as much context information as possible, since it will be taken into account at the next step to resolve to which devices firmware updates should be applied.

3) **Context-aware firmware assignment**: at this step, edge gateway will map target conditions of received firmware updates with the collected device context information, thus assigning new firmware to matching devices. In its simplest form, this can be implemented as a collection of AND and OR logical operators, where each IoT device is evaluated independently from the rest. More sophisticated scenarios might take into account the whole fleet of IoT devices, so that updates are assigned following some global policies. For example, it might be required to evenly distribute several firmware variants among the devices, or follow an A/B testing strategy, where some new code is tried only on a limited subset of devices. This step should also consider situations when an IoT device satisfies conditions of multiple updates, as well as when none suitable devices are found. As an outcome of this step, the deployment agent generates a map of firmware updates and matching IoT devices (assuming that a device can accept at most one update at time).

4) **Interaction with downstream IoT devices**: the firmware update pipeline concludes with the actual flashing of new code onto matching IoT devices. Once again, the collected context information about IoT devices will help the deployment agent to determine which underlying physical interface should be invoked in each individual case.

It is worth noting that despite the described ordered sequence, some steps can be executed in a circular and iterative manner. More specifically, it is possible that as a result of code deployment the context of IoT devices will change, thus possibly triggering another iteration of firmware assignment, and in some rare occasions lead to infinite loops or conflicts. Solving such emerging issues goes beyond the current paper, but is part of this overall research effort and is included as part of future work.

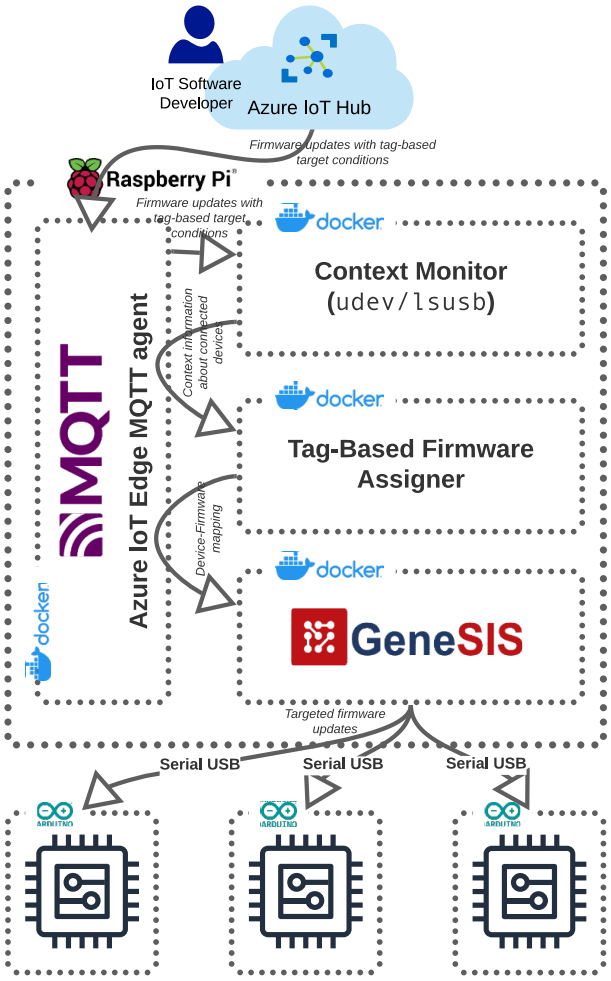

Fig. 2. Last-mile software deployment: proof of concept.

B. Proof of Concept

With the rapid adoption of edge computing, orchestration of containerised micro-services on edge gateways via a centralised cloud platform has recently become the state of practice, with the prominent offerings such as MS Azure IoT Edge,[3] AWS IoT Greengrass,[4] and Balena Cloud[5] available [21]. From a practical point of view, it is therefore a natural fit to implement the four-step functionality of the deployment agent as loosely coupled micro-services, packaged and provisioned as individual software containers, thus facilitating re-use, fault tolerance, and redundancy.

Accordingly, to implement the proof of concept prototype (see Fig. 2), we relied on Azure IoT Edge as the underlying cloud platform for *i)* installing the containerised deployment agent on edge devices, and *ii)* pushing firmware updates via the agent once it is up and running. The main reason behind this choice is the rich and mature support for scalable container management on edge gateways, as well as multiple services available through the Azure ecosystem. The latter factor was also aligned with our intention to re-use already existing tools whenever possible, instead of building from scratch or integrating third-party tools. Raspberry Pi 3 boards were used as edge gateways, while the role of terminal IoT devices was played by Arduino Uno boards. In our experimental setup, the Arduino boards were connected via the serial USB port.

The firmware deployment process is triggered whenever one or more code updates are released via the central IoT cloud platform. The software developer is expected to annotate each update with target conditions, i.e. tags determining to which IoT devices a given update should be loaded. As already mentioned, neither the software developer nor the cloud platform are aware of what particular devices will be affected by each update; this will be determined by edge gateways upon receiving the update commands. We now describe the implemented proof of concept using the previously outlined four steps:

1) **MQTT agent for interacting with the cloud**: Azure IoT Edge keeps track of the fleet of managed edge devices by continuously updating their digital twins. All communication between edge gateways and the cloud platform takes place

[3]https://azure.microsoft.com/services/iot-edge/
[4]https://aws.amazon.com/greengrass/
[5]https://www.balena.io/cloud/

via a standard MQTT API – lightweight and reliable solution for large-scale IoT systems. Among other things, device twins also report on the currently deployed and running software containers, as well as possible failures and errors. It is possible to selectively apply software deployment configurations to edge gateways using simple tag- and priority-based targeting. This way, the last-mile deployment agent can be installed only on a specific sub-set of the available edge gateways, e.g. within a specific geographical area or equipped with some required hardware. Accordingly, once up and running, the deployment agent starts continuously listening to incoming MQTT messages in order to further propagate firmware updates to target downstream IoT devices. Furthermore, the communication between the containers also takes place via MQTT.

2) **Context monitoring using standard Linux commands**: upon connecting a new IoT device to the gateway, it is then possible to query information about it using a combination of standard lsusb and udev Linux commands. Among other things, this yields information about the board model, MCU architecture, and port address. Although there is no direct way of querying the currently loaded sketch directly from an Arduino board, we can keep track of the loaded code on each device at the firmware deployment step. As a result, the edge gateway maintains a list of connected devices and associated context tags.

3) **Tag-based firmware assignment**: the current prototype implementation relies on a tag-based system for matching firmware updates to target IoT devices. Once a new firmware update is received from the cloud, the deployment agent evaluates the target conditions against the collected device context information to decide whether new code should be loaded. Tag-based assignment operates on individual devices, without taking into account the rest of the fleet. To enable more advanced assignment at the fleet level, we are also experimenting with the constraint solving [20] and combinatorial optimisation [23] techniques.

4) **GeneSIS for enacting firmware updates**: to implement the final step, we relied on our software deployment tool GeneSIS [24], [25]. GeneSIS is a toolkit for continuous orchestration and deployment of software components across the IoT-Edge-Cloud computing continuum. GeneSIS supports the automatic deployment within a local subsystem. When hosted on a local edge device, it can be used as a bridge to further deploy required code to associated downstream IoT devices. The GeneSIS modelling language is used to define software components, dependent libraries, and target devices, while the execution engine resolves what interface needs to be used to install or update the software artefacts accordingly. The GeneSIS engine is non-intrusive and does not require any bootstrap or pre-deployed agent running on a target IoT device to load code. Instead, it relies on an extensible plugin-based repository of supported IoT devices and interfaces. Currently, most of the Arduino-compatible boards are supported, and the available interfacing options are over-the-air updates using Bluetooth or Wi-Fi and serial communication over USB or direct wired connection using GPIO pins.

The described implementation of the deployment agent is one possible way of instantiating the proposed architecture for last-mile software deployment. This proof of concept has been validated in the context of a remote patient monitoring scenario [20], where sensor-enabled telecare devices are remotely re-programmed via a healthcare gateway installed on patients' premises. Nevertheless, the proposed approach can be successfully applied to other smart environments involving sensor-enabled IoT devices, edge gateways, and a centralised loud platform.

IV. Conclusion and Discussion

The proposed last-mile deployment agent, provisioned on edge gateways via the centralised cloud platform as containerised micro-services, is able to receive firmware updates from the cloud and apply them to connected IoT devices. The major benefit of this solution is that IoT software engineers are not required to know about target IoT devices beforehand when releasing firmware updates (which would be simply infeasible given the growing numbers). Instead, the task of assignment of firmware updates is shifted to the edge layer and takes place on gateways, which are able to evaluate target conditions against the collected cyber-physical context in order to selectively deploy new code to terminal IoT devices. By offloading the software assignment task to the edge layer, the proposed solution implements a hierarchical approach to software deployment in the IoT-Edge-Cloud continuum, and also contributes to the increased scalability by parallelising the software assignment task across multiple edge gateways, rather than on a centralised cloud. By integrating with the already available cloud-based container management at the edge, it is possible to build an end-to-end software management pipeline, so that even terminal IoT devices without Internet connectivity can be re-flashed in an automated scalable manner. This a big step towards a sustainable IoT where all the elements, even dormant and outdated, are not wasted, but are rather re-programmed and re-used.

A potentially promising direction for future work is to implement dynamic discovery of IoT devices in a local network of an edge gateway. The currently implemented prototype assumes that IoT devices are connected to the host gateway via a USB port; in this case, implementing a polling mechanism for continuously updating the list of connected devices and their cyber-physical contexts is relatively straight-forward using the standard Linux tools. A more challenging task would be to implement similar dynamic discovery in a local TCP/IP network or even via a wireless interface by detecting idle devices within the range of the edge gateway. Some initial attempts in this direction are reported in [26] and [27].

Another direction yet to be explored is the continuous self-adaptive behaviour of the system, as well as the intelligent resolution of infinite loops and potential conflicts between the changed context (caused by the applied firmware updates) and the target conditions. While the current proof of concept is implemented as a finite four-step pipeline triggered by the

initial update command received from the cloud, some advanced scenarios would require a continuous operation where firmware assignment and deployment would be triggered by the changing cyber-physical context of IoT devices.

While in this paper we focused on the sustainability problem of the exponentially growing IoT systems, the proposed solution has the potential to address a wide range of challenges related to re-use and re-purposing of legacy IoT devices – an increasingly pressing concern for modern smart and rapidly digitalised urban environments. There are obvious financial benefits for IoT software providers, who do not need to frequently invest in new infrastructure, and can also reduce operational costs related to manual device management. On the other hand, however, this can be seen as a threat to IoT hardware manufacturers whose profits are typically directly proportional to the number of manufactured and retailed units.

These economical considerations lead us to the final concluding remark. In this paper we focused on demonstrating only the technical feasibility of designing and implementing sustainable IoT systems for smart environments. There are, however, many other aspects to be considered and addressed in this respect. We intentionally omitted the discussion on the device ownership, access management, and security issues, which are major hindering factors. The proposed solution (and any other similar approaches relying on re-programming and re-purposing existing IoT infrastructures) is not yet ready to be immediately put into practice and adopted by the society. As any other 'green' and environmentally friendly initiative, it would require involving not just ICT researchers and IoT companies, but rather multiple stakeholders and policy makers from several adjacent domains, including governmental, environmental, and legal organisations, who should join their efforts to define common strategies for building sustainable IoT systems.

REFERENCES

[1] S. Higginbotham, "The internet of trash: IoT has a looming e-waste problem," *IEEE Spectrum: Technology, Engineering, and Science News*, vol. 17, 2018.

[2] V. Forti, C. P. Balde, R. Kuehr, and G. Bel, "The Global E-waste Monitor 2020: Quantities, flows and the circular economy potential," United Nations University/United Nations Institute for Training and Research, International Telecommunication Union, and International Solid Waste Association, Tech. Rep., 2020.

[3] B. Debnath, P. Roychowdhury, and R. Kundu, "Electronic Components (EC) reuse and recycling – a new approach towards WEEE management," *Procedia Environmental Sciences*, vol. 35, pp. 656–668, 2016.

[4] R. Arshad, S. Zahoor, M. A. Shah, A. Wahid, and H. Yu, "Green iot: An investigation on energy saving practices for 2020 and beyond," *IEEE Access*, vol. 5, pp. 15 667–15 681, 2017.

[5] L. Yin, H. Cheng, S. Mao, R. Haasch, Y. Liu, X. Xie, S.-W. Hwang, H. Jain, S.-K. Kang, Y. Su *et al.*, "Dissolvable metals for transient electronics," *Advanced Functional Materials*, vol. 24, no. 5, pp. 645–658, 2014.

[6] S. Yamada and H. Toshiyoshi, "A Water Dissolvable Electrolyte with an Ionic Liquid for Eco-Friendly Electronics," *Small*, vol. 14, no. 32, p. 1800937, 2018.

[7] P. Nguyen, N. Ferry, G. Erdogan, H. Song, S. Lavirotte, J.-Y. Tigli, and A. Solberg, "Advances in deployment and orchestration approaches for iot-a systematic review," in *2019 IEEE International Congress on Internet of Things (ICIOT)*. IEEE, 2019, pp. 53–60.

[8] W. McGrath, M. Etemadi, S. Roy, and B. Hartmann, "Fabryq: Using phones as gateways to prototype internet of things applications using web scripting," in *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 2015, pp. 164–173.

[9] N. Nikolov, "Research firmware update over the air from the cloud," in *2018 IEEE XXVII International Scientific Conference Electronics-ET*. IEEE, 2018, pp. 1–4.

[10] N. Jain, S. G. Mali, and S. Kulkarni, "Infield firmware update: Challenges and solutions," in *2016 International Conference on Communication and Signal Processing (ICCSP)*. IEEE, 2016, pp. 1232–1236.

[11] K. Abdelfadeel, T. Farrell, D. McDonald, and D. Pesch, "How to Make Firmware Updates over LoRaWAN Possible," in *2020 IEEE 21st International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE, 2020, pp. 16–25.

[12] A. Anastasiou, P. Christodoulou, K. Christodoulou, V. Vassiliou, and Z. Zinonos, "IoT Device Firmware Update over LoRa: The Blockchain Solution," in *2020 16th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, 2020, pp. 404–411.

[13] A. Kolehmainen, "Secure firmware updates for IoT: a survey," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2018, pp. 112–117.

[14] D. Wu, D. I. Arkhipov, E. Asmare, Z. Qin, and J. A. McCann, "UbiFlow: Mobility management in urban-scale software defined IoT," in *2015 IEEE conference on computer communications (INFOCOM)*. IEEE, 2015, pp. 208–216.

[15] J. Liu, Y. Li, M. Chen, W. Dong, and D. Jin, "Software-defined internet of things for smart urban sensing," *IEEE communications magazine*, vol. 53, no. 9, pp. 55–63, 2015.

[16] T. Ninikrishna, S. Sarkar, R. Tengshe, M. K. Jha, L. Sharma, V. Daliya, and S. K. Routray, "Software defined IoT: Issues and challenges," in *2017 International conference on computing methodologies and communication (ICCMC)*. IEEE, 2017, pp. 723–726.

[17] P. Mishra, D. Puthal, M. Tiwary, and S. P. Mohanty, "Software defined IoT systems: Properties, state of the art, and future research," *IEEE Wireless Communications*, vol. 26, no. 6, pp. 64–71, 2019.

[18] S. Nastic, S. Sehic, D.-H. Le, H.-L. Truong, and S. Dustdar, "Provisioning software-defined IoT cloud systems," in *2014 international conference on future internet of things and cloud*. IEEE, 2014, pp. 288–295.

[19] O. Bates and A. Friday, "Beyond data in the smart city: repurposing existing campus iot," *IEEE Pervasive Computing*, vol. 16, no. 2, pp. 54–60, 2017.

[20] H. Song, R. Dautov, N. Ferry, A. Solberg, and F. Fleurey, "Model-based fleet deployment of edge computing applications," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, 2020, pp. 132–142.

[21] R. Dautov and H. Song, "Towards Agile Management of Containerised Software at the Edge," in *2020 IEEE Conference on Industrial Cyber-physical Systems (ICPS)*, vol. 1. IEEE, 2020, pp. 263–268.

[22] R. Dautov and S. Distefano, "Distributed data fusion for the Internet of Things," in *International Conference on Parallel Computing Technologies*. Springer, 2017, pp. 427–432.

[23] R. Dautov, H. Song, and N. Ferry, "A light-weight approach to software assignment at the edge," in *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2020, pp. 380–385.

[24] N. Ferry, P. Nguyen, H. Song, P.-E. Novac, S. Lavirotte, J.-Y. Tigli, and A. Solberg, "Genesis: Continuous orchestration and deployment of smart iot systems," in *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1. IEEE, 2019, pp. 870–875.

[25] N. Ferry, P. H. Nguyen, H. Song, E. Rios, E. Iturbe, S. Martinez, and A. Rego, "Continuous deployment of trustworthy smart iot systems," *Journal of Object Technology*, vol. 19, no. 1, 2020.

[26] R. Dautov and S. Distefano, "Targeted content delivery to IoT devices using Bloom filters," in *International Conference on Ad-Hoc Networks and Wireless*. Springer, 2017, pp. 39–52.

[27] ——, "Stream Processing on Clustered Edge Devices," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2020.